**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CIS355A**

# Module 6: Read and Write to Databases

# Objectives

* Write data from the GUI to a database.
* Read data from the database and display the data on the GUI application.
* Use a three-tier architecture to handle large development projects.

We can read and write to files. However, what if your client has multiple locations? For example, your client may have an office in Atlanta, GA, another office in Dallas, TX, and another office in Los Angeles, CA. If you write your data to a file, the different locations will not be able to share the data. Even if your customer has a few locations in the same geographical area, the nearby locations cannot share information with each other if you write the data to a file. How is it possible for huge companies to share their information with all their world-wide locations? They write the information to databases. In fact, you can put your database in the cloud and have all of your locations read and write to that single database.

Databases are so important and so popular that many people make an entire career out of databases. Google “department of labor database administrator” and you will see that database administrators make a very good living.

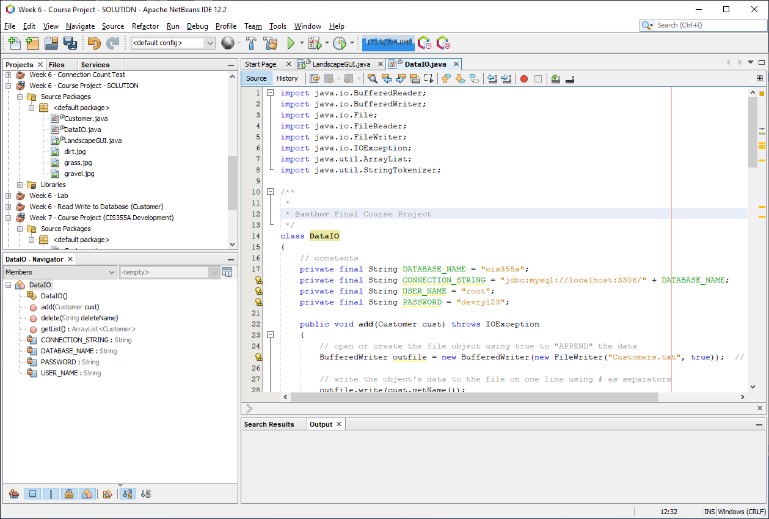
Most industry applications read and write to databases. As a software developer, it is important to know how to use databases and how to read and write to them. Let’s update our application so it can read and write to a database.

Remember that we created our application using a three-tier architecture.

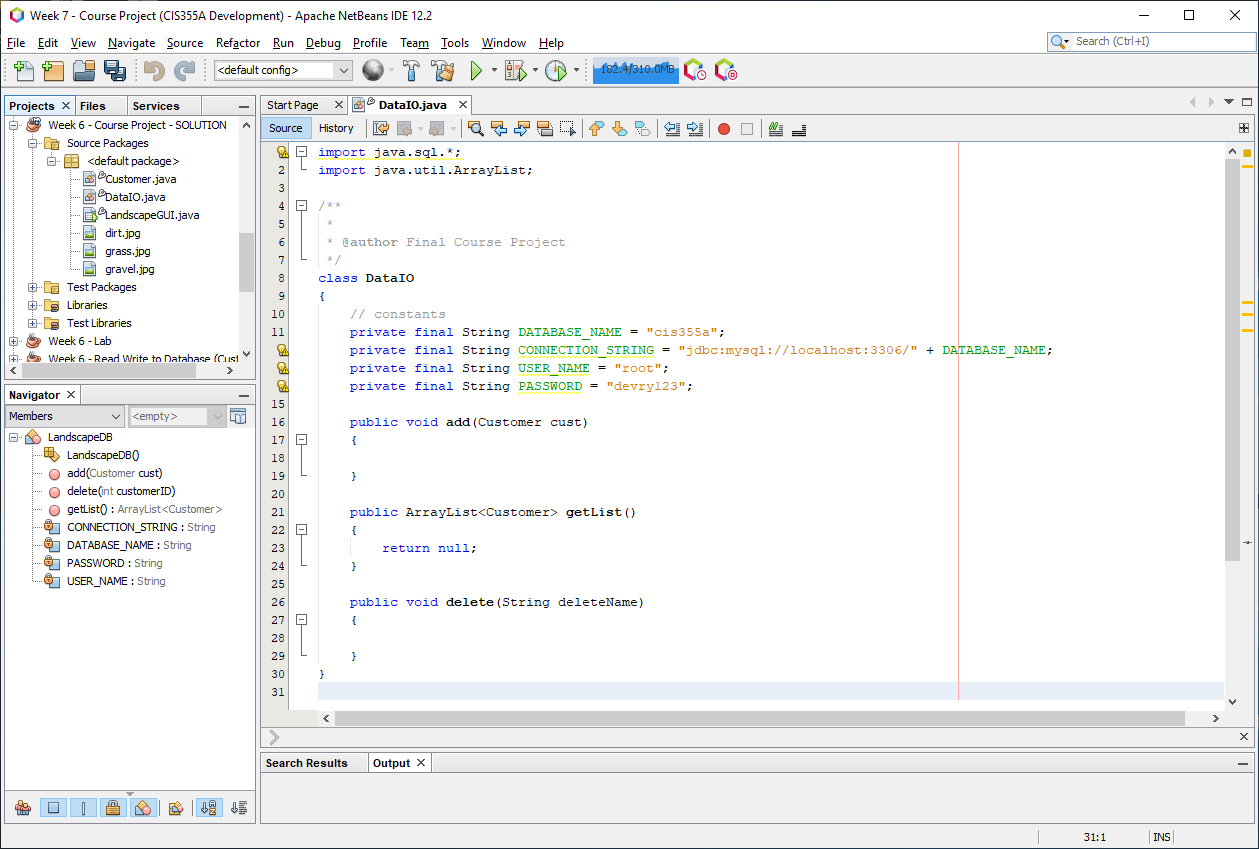
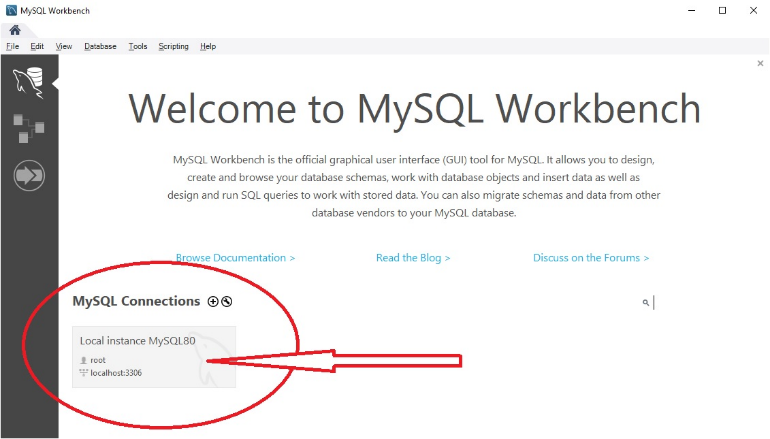
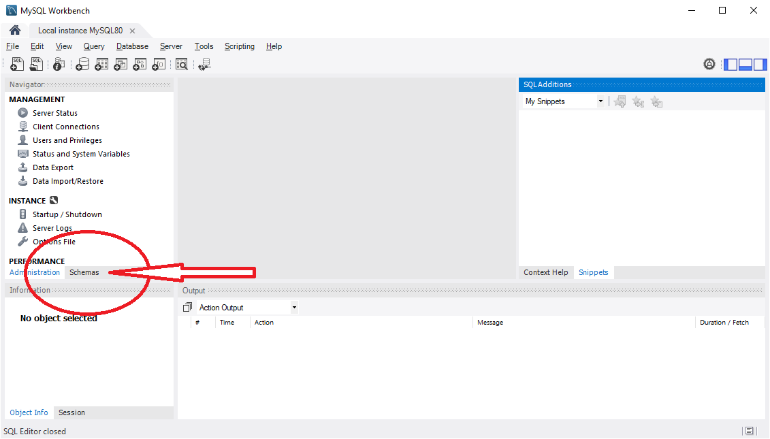
* Presentation: GUI classes, which read info from the user and displays info to the user
* Business rules: handles data processing and class organization for the application
* Data storage: use a separate class to read and write to the file or database

We get to benefit from this design style this week. Most of our work will be in the data storage tier. You are going to love it.

# Steps

1. Before we can read or write to a database, we need to have the database driver in the root folder of our project. You can get the database driver from your instructor or you can Google for it and download it from the database company. Because we are using MySQL, Google “mysql java driver”. It may change, but my top search result is “MySQL Connector/J” located at <https://www.mysql.com/products/connector/>. Then, I click on the download link next to JDBC Driver for MySQL. I choose the operating system as Platform Independent and then click on the download for the Zip Archive. On the next page, I click the link “No thanks, just start my download.” Once the file downloads, I extract it and find the mysql-connector-java-*xxxx*.jar file (xxxx part will change). This is the only file that you will need.
2. Open your project’s folder. Copy the mysql-connector-java-*xxxx*.jar file to the root folder (base folder). Once the file is there, open your NetBeans project. Click the + sign next to the project name. Then, right-click on Libraries. Choose Add Jar/Folder. Browse and find your project’s folder. Once you find the folder, you will see the mysql-connector-java-*xxxx*.jar file listed. Double-click on the file or select it and then click Open to add the MySQL driver to your project library.
3. Open your Visio Class diagram from our design phase. We need to update the DataIO class to match our Visio Class diagram. Remember, DataIO stands for data input and output. Because we are connecting with a database, we will need to add our connection variables as constants to the top of the class.  
     
   
4. If you made the password devry123 and called your schema cis355a, then your constants should be as follows.
5. private final String DATABASE\_NAME = "cis355a";  
   private final String CONNECTION\_STRING =   
    "jdbc:mysql://localhost:3306/" + DATABASE\_NAME;  
   private final String USER\_NAME = "root";  
   private final String PASSWORD = "devry123";  
     
   Once you finish updating the DataIO class, the top of the class should look something like this.  
     
   
6. The DataIO class is set up to read and write to files. Let’s update it so it reads and writes to the database. Delete the import statements at the top and replace them with import statements for database access. The asterisk (‘\*’) tells Java to import all classes that are referenced from the java.sql package. We also need to return an ArrayList, so let’s import that class as well from the java.util package.  
     
   import java.sql.\*;

import java.util.ArrayList;

1. Delete the code in your methods so we can add the database input and output code, including the throws statements at the end of the method headers. When you finish, your DataIO border class should look something like this.  
     
   
2. Now, we need to set up our MySQL database so it can hold our data. Open your MySQL workbench. Click on the local instance MySQL80 icon.  
     
   
3. Type in your password, which should be devry123 (without the quotes) if you set it up following the default directions. Otherwise, please enter your chosen password. Click on the schemas tab so you can see your databases, which are called Schemas in MySQL.  
     
   
4. You should have a database (schema) called cis355a. If you do not have one, click the barrel icon (“Create new schema in the connected server”) that is located on the Taskbar at the top to create it. Open the cis355a database by clicking the triangle next to it. Open the tables by clicking the triangle next to it so you can see the existing tables. Right-click on tables and choose Create Table. Create the following table and call it landscape. Notice that CustomerID is a primary key and it has Auto Increment turned on.  
     
   
5. Go back to your DataIO class. In the add( ) method, write code to do the following.
   1. Check for the database driver.
   2. Connect to the database.
   3. Add the Customer record to the landscape table.
   4. Close the database connection.

Remember to throw the exceptions so the GUI class can report any possible issues. Also, remember to use a PreparedStatement to help prevent hacking. When you finish, your code should look something like this.

public void add(Customer cust) throws ClassNotFoundException, SQLException

{

//check for driver

Class.forName("com.mysql.cj.jdbc.Driver");

//connect to database

Connection conn = DriverManager.getConnection(CONNECTION\_STRING,

USER\_NAME, PASSWORD);

//add record

String strSQL = "INSERT INTO landscape (CustomerName, CustomerAddress, "

+ "LandscapeType, YardLength, YardWidth, LandscapeCost) "

+ "VALUES(?, ?, ?, ?, ?, ?)";

PreparedStatement pstmt = conn.prepareStatement(strSQL);

pstmt.setString(1, cust.getName());

pstmt.setString(2, cust.getAddress());

pstmt.setString(3, cust.getYardType());

pstmt.setDouble(4, cust.getLength());

pstmt.setDouble(5, cust.getWidth());

pstmt.setDouble(6, cust.getTotalCost());

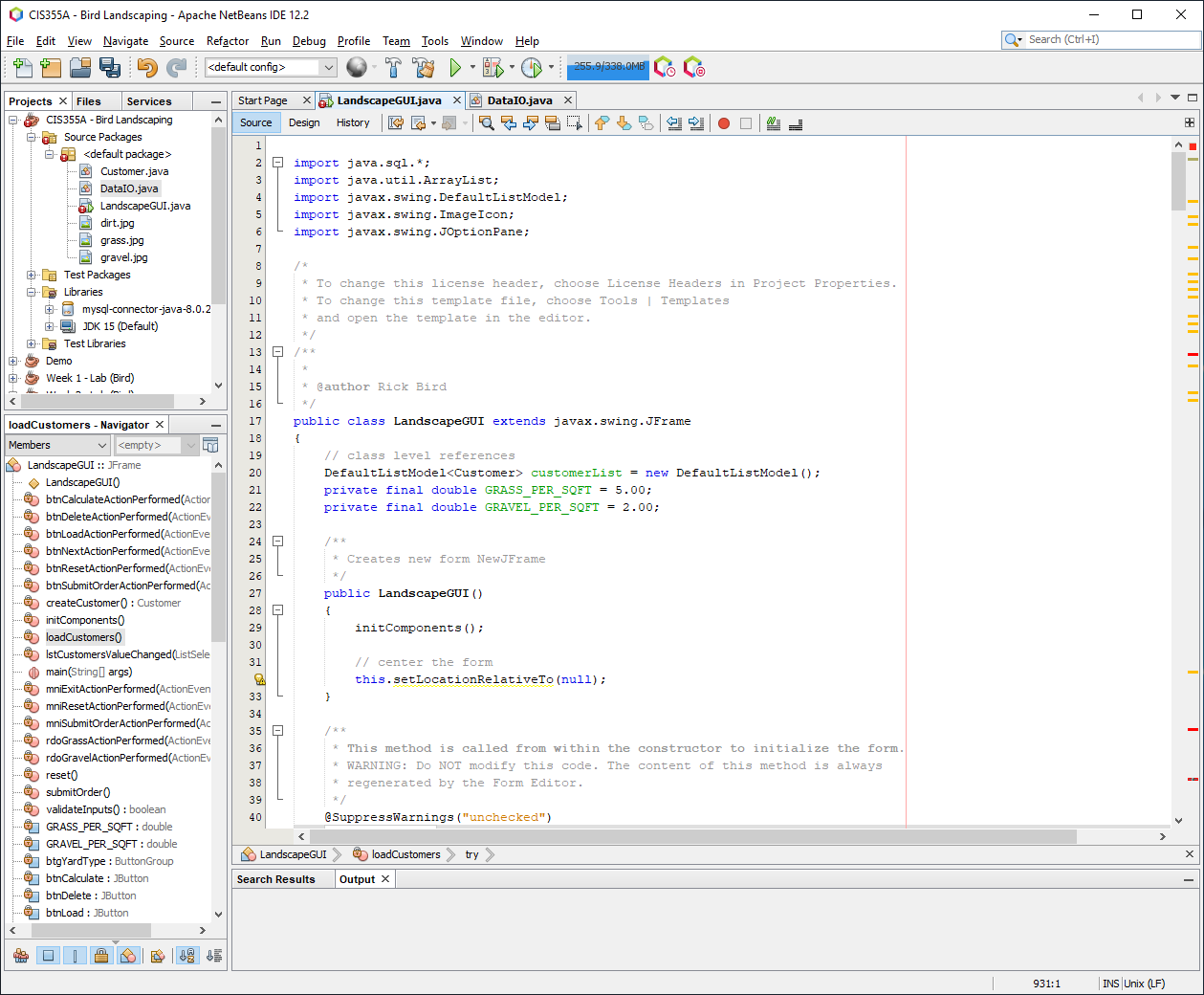
// execute the prepared statement

pstmt.execute();

//close connection

conn.close();

}

1. Go to your LandscapeGUI.java file. Go to the top of your code window and replace the IOException import with an import for java.sql.\*. When you finish, the top of your code should look like this.  
     
   
2. Update you submitOrder( ) method so that it looks for SQLExceptions instead of IOExceptions. In addition, we need to look for the ClassNotFoundException in case we do not have the database driver installed correctly. When you finish, the submitOrder( ) method should look something like this.

private void submitOrder()

{

if (validateInputs() == false)

{

return; // end the method if validation failed

}

Customer cust = createCustomer();

customerList.addElement(cust);

txaOrderInfo.setText(cust.getDetails());

try

{

DataIO data = new DataIO(); // create DataIO object

data.add(cust);

loadCustomers(); // load all customers

// reset for the next customer

reset();

//move to the client orders tab

tabMain.setSelectedIndex(2);

}

catch (SQLException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"Database Error", JOptionPane.ERROR\_MESSAGE);

}

catch (ClassNotFoundException ex)

{

JOptionPane.showMessageDialog(this, "Driver Not Found Error: " + ex.getMessage(),

"Database Driver Error", JOptionPane.ERROR\_MESSAGE);

}

}

1. If you use a block comment on your btnDeleteActionPerformed( ) method and your loadCustomers( ) method, you can test your submitOrder( ) method. For example, put a \\* as the first line of your loadCustomers method and a \*\ as the last line.

private void loadCustomers()

{

/\*

// the code will be “commented out” and ignored by Java

\*/

}

1. Go to your DataIO class. In order to read data from the database, we need to connect to the database, read the records, and then close the connection. The file reads the data using a while loop, and then close the file. As we read the records, let’s create Customer objects and then add them to the ArrayList. Finally, we need to return the ArrayList. When you finish, your code should look like something like this (including the throws statement).

public ArrayList<Customer> getList() throws SQLException

{

// create the ArrayList so we have something to return

ArrayList<Customer> list = new ArrayList<Customer>();

//connect to database

Connection conn = DriverManager.getConnection(CONNECTION\_STRING,

USER\_NAME, PASSWORD);

Statement statement = conn.createStatement();

String SQL = "Select \* FROM landscape";

ResultSet rs = statement.executeQuery(SQL);

while (rs.next())

{

// create Customer object and load the attributes

Customer client = new Customer();

client.setCustomerID(rs.getInt(1));

client.setName(rs.getString(2));

client.setAddress(rs.getString(3));

client.setYardType(rs.getString(4));

client.setLength(rs.getDouble(5));

client.setWidth(rs.getDouble(6));

client.setTotalCost(rs.getDouble(7));

// add the Customer object to our list

list.add(client); }

// close the database connection

conn.close();

// return the ArrayList

return list;

}

1. Save All and then go to your LandscapeGUI.java file. Update your loadCustomers( ) method so that it looks for SQLExceptions instead of IOExceptions. When you finish, the loadCustomers( ) method should look something like this.

private void loadCustomers()

{

try

{

DataIO data = new DataIO(); // create DataIO object

ArrayList<Customer> customers = data.getList();

// clear out the DefaultListModel and textarea

customerList.clear();

txaOrderInfo.setText("");

// copy each object from the ArrayList over to the DefaultListModel

for (int i = 0; i < customers.size(); i++)

{

customerList.addElement(customers.get(i));

}

}

catch (SQLException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"Database Error", JOptionPane.ERROR\_MESSAGE);

}

}

1. Run your application. Click on the customer list tab. Click on the load list button. Does it work? So far, so good.
2. We need to be able to delete a customer. Go to your DataIO class. Change the method header so we can delete the customer based on the customer’s ID number. In your delete( ) method, connect to the database. Then, issue the DELETE FROM command. Finally, close the connection to the database. When you finish, the delete( ) method should look something like this.

public void delete(int customerID) throws SQLException

{

// connect to the database

Connection conn = DriverManager.getConnection(CONNECTION\_STRING,

USER\_NAME, PASSWORD);

// delete the record

String SQL = "DELETE FROM landScape WHERE CustomerID = ?";

PreparedStatement pstmt = conn.prepareStatement(SQL);

pstmt.setInt(1, customerID);

pstmt.execute();

// close the database connection

conn.close(); }

}

1. Let’s update our delete customer button event code in the GUI. Save All and then go to the code and get the selected customer object. Then, create a DataIO object and delete the customer based on the customer’s ID number. Finally, load the current customers using the loadCustomers( ) method. When you finish, your code should look something like this.

try

{

// get the selected object

Customer old = lstCustomers.getSelectedValue();

// if something is selected, delete it and clear the details textarea

if (old != null)

{

DataIO data = new DataIO();

data.delete(old.getCustomerID()); // get the name only

txaCustomerInfo.setText("");

loadCustomers();

}

}

catch (SQLException ex)

{

JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(),

"Database Error", JOptionPane.ERROR\_MESSAGE);

}

1. Run your application and test it, including the menu. You can now read and write to databases. This is an amazing project. What projects will you build now that you can design a GUI and read/write to database? The sky is the limit.

# Course Project Deliverables for Week 6

* Close NetBeans so your zip file will have the latest code. Then, go to the project folder. Right-click on the project folder and choose Send to 🡪 Compressed (zipped) file. Then, rename the zipped file to include your name. Submit the zipped file on Canvas.
  + Week 6 Database IO – Your Name.zip